計算機アーキテクチャ 97－5
（1992. 12. 3）

メモリ・コンシステンシ・モデル
— フレームワーク化および新モデルの提案 —

徳永 尚哉　　村上 和彰

九州大学 大学院総合理工学研究科

マルチプロセッサのプロセッサ間通信モデルとして共有メモリ・モデルを用いる場合,「メモリ・コンシステンシ（*memory consistency*）を如何に保つか」，すなわち,「メモリ・コンシステンシ・モデル（*memory consistency model*）」の設定が重要な課題となる．これまでに，いくつかのメモリ・コンシステンシ・モデルが提案されているが，その定義および用法には少々混乱が見られる．そこで，本稿では，これらモデルの理解を容易にし，かつ，共通の土台でモデル間の比較が可能となるように，メモリ・コンシステンシ・モデルに対して統一的枠組（フレームワーク）を与えている．そして，このフレームワークに基づいて，従来から提案されてるメモリ・コンシステンシ・モデルを再定義するとともに，新しいメモリ・コンシステンシ・モデルを4つ提案している．本稿で提案するモデルは，従来のモデル（*Weak Consistency*, *Release Consistency*, および, *Data-Race-Free-1*）における「ハードウェアが遵守すべき条件」のいくつかの条件を緩和して，モデルの潜在能力を向上させたものである．

Memory Consistency Models
— Frameworking and Proposing New Models —

Naoya TOKUNAGA　Kazuaki MURAKAMI

Department of Information Systems
Interdiciplinary Graduate School of Engineering Sciences
Kyushu University
6-1 Kasuga-koen, Kasuga-shi, Fukuoka 816 JAPAN
E-mail:{*tokunaga, murakami*}@is.kyushu-u.ac.jp

The performance of a shared-memory multiprocessor depends largely on the choice of the memory consistency model supported. Several different consistency models have been proposed in the literature. Unfortunately, due to the lack of common frameworks, there are some fallacies and confusions on those consistency models. This paper thus tries to establish a common framework on memory consistency models to make the models easy to understand and compare. According to the proposed framework, the paper give rigorous definitions to previously proposed consistency models. The paper then attempts to improve potential performance attainable by some models, and results in proposing four new consistency models.

1 はじめに

マルチプロセッサのプロセッサ間通信モデルとして，共有メモリ（*shared-memory*）モデルとメッセージ交換（*message-passing*）モデルの2つが存在する．このうち，共有メモリ・モデルにおいては，「メモリ・コンシステンシ（*memory consistency*：一貫性）を如何に保つか」，すなわち，「メモリ・コンシステンシ・モデル（*memory consistency model*）」の設定が重要な課題となる．これまでに，以下のような種々のメモリ・コンシステンシ・モデルが提案されているが，その定義および用法には少々混乱が見られる．

① *Sequential Consistency* (*SC*)[10]
② *Processor Consistency* (*PC*)[9]
③ *Weak Consistency* (*WC*)[6]
④ *Data-Race-Free-0* (*DRF0*)[4]
⑤ *Release Consistency* (*RC*)[7]
⑥ *Data-Race-Free-1* (*DRF1*)[5]

そこで，我々は，これらモデルの理解を容易にし，かつ，共通の土台でモデル間の比較が可能となるように，メモリ・コンシステンシ・モデルに対して統一的枠組（フレームワーク）を与えた[1, 2]．本稿では，このフレームワークに基づいて，従来から提案されてるメモリ・コンシステンシ・モデルを再定義するとともに，新しいメモリ・コンシステンシ・モデルを提案する．

まず，2章で本稿で用いる用語を定義した後，3章でメモリ・コンシステンシ・モデルのフレームワークを与える．なお，このフレームワークは，先に文献[1]で定義したものを改訂したものである．次に，4章で，上記の6モデル中②を除く5モデルに対して，本フレームワークを適用する[1]．そして，5章で新しいメモリ・コンシステンシ・モデルを提案する．最後に，6章で今後の課題をまとめる．

2 準備

2.1 対象

次の条件を満たす共有メモリ型マルチプロセッサを議論の対象とする．

- 各プロセッサはその識別子により一意に特定可能である．
- 共有メモリはグローバルなアドレスを有し，また，すべてのプロセッサは共有メモリに関して同一のアドレス空間を有する．

2.2 メモリ・アクセスとその完了

定義 1 (メモリ・アクセス) [2]
メモリ・アクセスのインスタンスは，以下の4つ組で与えられる．

$$M = (P, N, T, L)$$

ここで，各成分は以下のものとする．

- P: 当該メモリ・アクセスを行うプロセッサの識別子．
- N: 当該メモリ・アクセスが，プロセッサ Pにおいて何番目のインスタンスか．
- T: load，store等のメモリ・アクセスの種類．
- L: 当該メモリ・アクセスが行われるメモリ上の位置（アドレス）．

定義 2 (メモリ・アクセスの完了) [6]
プロセッサ P_1の行ったメモリ・アクセス $M_1 = (P_1, N_1, T_1, L_1)$ は，以下の条件を満足した時点で「プロセッサ P_2 に関して完了した」と言う．

- T_1 =ロードの場合：プロセッサ P_2が L_1 に対してストア・アクセスを行っても P_1に返る値に影響しない．
- T_1 =ストアの場合：プロセッサ P_2が L_1 に対してロード・アクセスを行ったら，M_1のストア値を P_2に返す．

全プロセッサに関して完了した時点で「メモリ・アクセス M_1は完了した」と言う．

2.3 メモリ・アクセス集合およびオーダー

定義 3 (メモリ・アクセス集合)
プログラムの実行によって生じたすべてのメモリ・アクセスを要素とする有限集合をメモリ・アクセス集合と呼ぶ．

定義 4 (オーダー（order）) [2]
メモリ・アクセス集合上の半順序関係において，2つの要素 M_1と M_2が $M_1 < M_2$である時，$M_1 \xrightarrow{order} M_2$と表記する．この「$\xrightarrow{order}$」を「$M_1$から M_2へのオーダー」と呼ぶ．

2.4 衝突と競合

定義 5 (衝突対) [11]
メモリ・アクセス集合の2つの要素 $M_1 = (P_1, N_1, T_1, L_1)$ と $M_2 = (P_2, N_2, T_2, L_2)$ が以下の条件を満たす時，「M_1と M_2は衝突対（*conflicting pair*）を成す」と言う．

- $L_1 = L_2$
- T_1 =ストア，または，T_2 =ストア
- $P_1 \neq P_2$，または，$N_1 \neq N_2$

定義 6 (競合対) [7]
メモリ・アクセス集合の2つの要素 M_1と M_2が以下の条件を満たす時，「M_1と M_2は競合対（*competing pair*）を成す」と言う[2]．

- M_1と M_2が衝突対を成す．
- M_1と M_2との間にオーダーが存在しない．すなわち，$M_1 \xrightarrow{order} M_2$，あるいは，$M_2 \xrightarrow{order} M_1$のいずれでもない．

3 フレームワーク

我々は，メモリ・コンシステンシ・モデルをソフトウェアとハードウェアとの間の「取り決め」と見る（図

[1] *PC*を除いたのは，文献[1]で示した*SC*クラスに*PC*が属さないからである．

[2] 文献[7]の定義とは異なり，同一プロセッサから出されたメモリ・アクセス同士でも競合対を成し得る．

1参照)．この「取り決め」に従って，プログラムを記述し，かつ，ハードウェアが当該プログラムを実行した場合に，メモリ・コンシステンシが保証される．

定義 7 (コンシステンシ・モデル)
コンシステンシ・モデル CM は，次の4つ組で与えられる．

$$CM = (I, O, P_{SW}, P_{HW})$$

ここで，各成分は以下のものとする．

- I: 命令セットとして定義されたメモリ・アクセス命令を要素とする集合．
- O: メモリ・アクセス間に存在し得るオーダーを要素とする集合．
- P_{SW}: ソフトウェア（プログラム）が遵守すべき条件を要素とする集合．
- P_{HW}: ハードウェアが遵守すべき条件を要素とする集合．

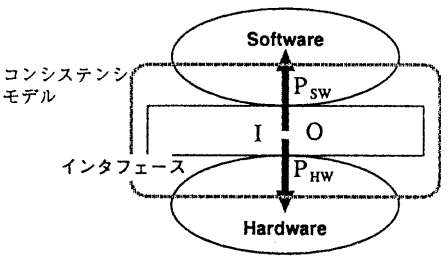

図 1: メモリ・コンシステンシ・モデル

4 従来モデルの再定義

3章で与えたフレームワークを従来の5つのメモリ・コンシステンシ・モデル（*SC*, *WC*, *DRF0*, *RC*, *DRF1*）に適用した結果を表1に示す．
以下，表中に現れる記号の定義を与える．

4.1 メモリ・アクセス命令

表1の I欄に現れるのは，以下のメモリ・アクセス命令である．

- load：同期変数以外に対するロード・アクセス．
- store：同期変数以外に対するストア・アクセス．
- special：同期変数へのアクセス．これはさらに，以下のように区別することもある．
  - acquire：クリティカル・セクションの入口における，同期変数（ロック変数）へのロード・アクセス．
  - release：クリティカル・セクションの出口における，同期変数（ロック変数）へのストア・アクセス．
  - nsync：上記以外の（つまり，クリティカル・セクションに無関係な）同期変数へのアクセス．

4.2 オーダー

表1の O欄に現れるのは，以下のオーダーである．

- $\xrightarrow{po0}$ (*program-order-0*)：メモリ・アクセス集合の2つの要素 $M_1 = (P_1, N_1, T_1, L_1)$ と $M_2 = (P_2, N_2, T_2, L_2)$ が以下の条件を満たす時，および，その時に限り M_1 と M_2の間に $M_1 \xrightarrow{po0} M_2$ なるオーダー「$\xrightarrow{po0}$」が存在する．
  - $P_1 = P_2$
  - $N_1 < N_2$
- $\xrightarrow{so0}$ (*special-order-0*)[3]：メモリ・アクセス集合の2つの要素 $M_1 = (P_1, N_1, T_1, L_1)$ と $M_2 = (P_2, N_2, T_2, L_2)$ が以下の条件を満たす時，および，その時に限り M_1 と M_2の間に $M_1 \xrightarrow{so0} M_2$ なるオーダー「$\xrightarrow{so0}$」が存在する．
  - T_1 = special
  - T_2 = special
  - $L_1 = L_2$
  - M_1が M_2より先に完了する．
- $\xrightarrow{so1}$ (*special-order-1*)[4]：メモリ・アクセス集合の2つの要素 $M_1 = (P_1, N_1, T_1, L_1)$ と $M_2 = (P_2, N_2, T_2, L_2)$ が以下の条件を満たす時，および，その時に限り M_1 と M_2の間に $M_1 \xrightarrow{so1} M_2$ なるオーダー「$\xrightarrow{so1}$」が存在する．
  - T_1 = release
  - T_2 = acquire
  - $L_1 = L_2$
  - M_2が M_1によってストアされた値をロードする．
- $\xrightarrow{so2}$ (*special-order-2*)：メモリ・アクセス集合の2つの要素 $M_1 = (P_1, N_1, T_1, L_1)$ と $M_2 = (P_2, N_2, T_2, L_2)$ が以下の条件を満たす時，および，その時に限り M_1 と M_2の間に $M_1 \xrightarrow{so2} M_2$ なるオーダー「$\xrightarrow{so2}$」が存在する．
  - T_1 = release
  - T_2 = acquire
  - $M_1(\xrightarrow{po0} \cup \xrightarrow{so0})^+ M_2$
- $\xrightarrow{pso0}$ (*program-special-order-0*)[5]：$\xrightarrow{po0}$と$\xrightarrow{so0}$の非反射的推移閉包，すなわち，

$$\xrightarrow{pso0} = (\xrightarrow{po0} \cup \xrightarrow{so0})^+$$

- $\xrightarrow{pso1}$ (*program-special-order-1*)[6]：$\xrightarrow{po0}$と$\xrightarrow{so1}$の非反射的推移閉包，すなわち，

$$\xrightarrow{pso1} = (\xrightarrow{po0} \cup \xrightarrow{so1})^+$$

- $\xrightarrow{pso2}$ (*program-special-order-2*)[7]：$\xrightarrow{po0}$と$\xrightarrow{so2}$の非反射的推移閉包，すなわち，

$$\xrightarrow{pso2} = (\xrightarrow{po0} \cup \xrightarrow{so2})^+$$

[3]文献 [4] では，synchronization-order-0 と呼んでいる．
[4]文献 [5] では，synchronozation-order-1 と呼んでいる．
[5]文献 [4] では，happens-before と呼んでいる．
[6]文献 [5] では，happens-before-1 と呼んでいる．
[7]文献 [7] では，呼び名を与えていない．

4.3 ソフトウェアが遵守すべき条件

表1の P_{SW} 欄に現れるのは，以下の条件である．

- 条件1：当該プログラムを実行した際に生じ得るすべての衝突対のうち，プロセッサが同一のもの（$P_1 = P_2$）については，その衝突対を成す2つのメモリ・アクセス間に O の要素であるいずれかのオーダーが存在していなければならない．
- 条件2：当該プログラムを実行した際に，競合対が存在してはいけない．すなわち，当該プログラムを実行した際に生じ得るすべての衝突対について，その衝突対を成す2つのメモリ・アクセス間に O の要素であるいずれかのオーダーが存在していなければならない．

4.4 ハードウェアが遵守すべき条件

表1の P_{HW} 欄に現れるのは，以下の条件である．

- 条件A — プロセッサ内データ依存関係（*intraprocessor data dependency*）の保証：
  - 条件A-0：プロセッサ内データ依存関係を保証する．すなわち，$M_1 \xrightarrow{po0} M_2$ なる衝突対について，M_2 をプロセッサ P_2（$= P_1$）に関して完了させるためには，M_1 がプロセッサ P_1（$= P_2$）に関して完了していなければならない．
- 条件B — メモリ・アクセスを実行したプロセッサが当該メモリ・アクセスを完了させるための条件：
  - 条件B-0 (*Intraprocessor, Ordinary → Ordinary*)：あるプロセッサが実行したload/storeを完了させるためには，当該プロセッサがそれ以前に実行したすべてのload/storeが完了していなければならない．
  - 条件B-1 (*Intraprocessor, Special → Ordinary*)：あるプロセッサが実行したload/storeを完了させるためには，当該プロセッサがそれ以前に実行したすべてのspecialが完了していなければならない．
  - 条件B-2 (*Intraprocessor, Ordinary → Special*)：あるプロセッサが実行したspecialを完了させるためには，当該プロセッサがそれ以前に実行したすべてのload/storeが完了していなければならない．
  - 条件B-3 (*Intraprocessor, Special → Special*)：あるプロセッサが実行したspecialを完了させるためには，当該プロセッサがそれ以前に実行したすべてのspecialが完了していなければならない．
  - 条件B-4 (*Intraprocessor, Special → Ordinary, Eager*)：あるプロセッサが実行したload/storeを完了させるためには，当該プロセッサがそれ以前に実行したすべてのspecialがあるプロセッサに関してそれぞれ完了していなければならない．
  - 条件B-5 (*Interprocessor, Ordinary → Special*)：あるプロセッサが実行した（$M_1 \xrightarrow{so0} M_2$ なる衝突対を成す）special$M_2 = (P_2, N_2, T_2, L2)$ を完了させるためには，相手プロセッサ $P_1 (\neq P_2)$ が special$M_1 = (P_1, N_1, T_1, L1)$ 以前に実行したすべてのload/storeが完了していなければならない．
  - 条件B-6 (*Intraprocessor, Acquire → Ordinary*)：あるプロセッサが実行したload/storeを完了させるためには，当該プロセッサがそれ以前に実行したすべてのacquireが完了していなければならない．
  - 条件B-7 (*Intraprocessor, Ordinary → Release*)：あるプロセッサが実行したreleaseを完了させるためには，当該プロセッサがそれ以前に実行したすべてのload/storeが完了していなければならない．
  - 条件B-8 (*Interprocessor, Special → Special*)：specialアクセス M_1 のあるプロセッサ P_i に関する完了が，specialアクセス M_2 のあるプロセッサ P_j に関する完了よりも先行する場合，M_2 を完了させるためには M_1 が完了していなければならない．
- 条件C — メモリ・アクセス完了の順序に関する条件：
  - 条件C-0：衝突対を成すstoreアクセスは，すべてのプロセッサに関して同一順序で完了しなければならない．
  - 条件C-1：衝突対を成すspecialアクセスは，すべてのプロセッサに関して同一順序で完了しなければならない．
  - 条件C-2：$M_1 \xrightarrow{pso1} M_2$ なる衝突対を成すメモリ・アクセス M_1 および M_2 は，すべてのプロセッサに関して M_1，M_2 の順序で完了しなければならない．

5 新しいモデルの提案

4章で示した従来のメモリ・コンシステンシ・モデルを，次の2つのアプローチにより改良する．

- *WC*, *DRF0*, *RC*, および，*DRF1* に対する P_{HW} の条件A-0の緩和．
- さらに，*RC* に対する P_{HW} の条件B-6&7の緩和．

5.1 条件A-0の緩和

SC を除く4つのモデルはすべて，P_{HW} として条件A-0を含んでいる[8]．すなわち，プロセッサ内データ依存関係をハードウェアにより保証しなければならない．これを性能を低下させずに実現するには，未完了メモリ・アクセスを保持するバッファ（たとえば，ストア・バッファ）に対してアドレス（たとえば，ロード・アドレス）でもって連想アクセスが行える必要がある．そして，一致するアドレス（たとえば，ストア・アドレス）が存在する場合，対応するデータ（たとえば，ストア・データ）を（たとえば，ロード・データとして）フォワーディングできなければならない．結果的に，大量の比較器およびバイパス回路がハードウェアとして必要となる．

しかしながら，メモリ・オペランドに関してデータ依存関係にあるようなメモリ・アクセス（たとえば，同一アドレスに対するストアとロード）が，短時間に連続して実行されることは稀である．このような稀な

[8] *SC* は，陽には条件A-0を含まないが，条件B-0により条件A-0を満たしている．

ケースに対して性能を維持するために上記のような大量のハードウェアを費やすのは，コスト／パフォーマンスの点で効率が悪い．そこで，3モデルに対して，「性能をまったく低下させずに，プロセッサ内データ依存関係をハードウェアで保証する」というアプローチを捨てることを検討してみよう．

代替アプローチとしては，次の2つが考えられる．

第1のアプローチ プロセッサ内データ依存関係をハードウェアではまったく保証しない．ストア・バッファ等には比較器はまったく不要となる．もし，プロセッサ内データ依存関係を保証しなければならない場合は，プロセッサ間データ依存関係を保証するのと同じようにクリティカル・セクションを用いる（図2(a)参照）[9]．これは，1個のプロセッサ上で複数のプロセス（あるいは，スレッド）を実行するような環境では，ごく自然に起こり得る状況である．たとえば，クリティカル・セクションを用いて共有データにアクセスするプロセスが複数個たまたま同一プロセッサ上で実行されたとしよう．このとき，当該共有データに関してプロセッサ内データ依存関係が生じるが，これをハードウェアで保証する必要はない．なぜなら，当該データ依存関係はクリティカル・セクションによって保証されているからである．しかしながら，1個のプロセス内の占有データへのメモリ・アクセスの度にクリティカル・セクションを用いるようになると，性能低下が大きくなる．

第2のアプローチ 明示されたプロセッサ内データ依存関係はハードウェアで保証するが，この際若干性能が低下する．ストア・バッファ等に連想アクセス機能は必要ないが，比較器が少なくとも1個は必要である．これには，次の2種類のload/storeを定義する（図2(a)参照）．

- IDload/ IDstore(*Independent* load/store)：プロセッサ内データ依存関係を保証しない．たとえば，IDloadを実行する場合，ストア・バッファを検索したりしない．
- Dload/ Dstore(*Dependent* load/store)：プロセッサ内データ依存関係を保証する．たとえば，IDloadを実行する場合ストア・バッファを検索するが，これは1個の比較器を用いて逐次的に行う．

本稿では，第2のアプローチを *WC*, *RC*, および，*DRF1* に対して適用して得られる，以下の3つのメモリ・コンシステンシ・モデルを新たに提案する．

① *Improved Weak Consistency* (*IWC*)
② *Improved Release Consistency* (*IRC*)
③ *Improved Data-Race-Free-1* (*IDRF1*)

5.2 条件B-6&7の緩和

RC（ないし *DRF1*）における acquire と release は，クリティカル・セクションのそれぞれ入口と出口に対応しており，機能的には以下の役目を持つ．

[9] *DRF0* には適用できない．また，*WC* の場合，データ依存関係にある命令間に special を挿入するだけで十分である．

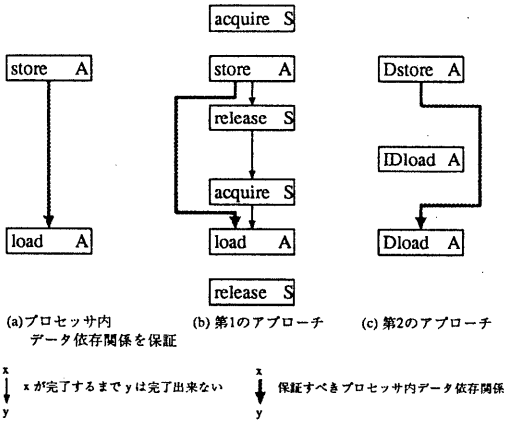

図 2: 条件 A-0 の緩和

- release：(クリティカル・セクション内のメモリ・アクセスか否かに関わらず) 先行するすべてのメモリ・アクセスが完了したことを，acquire を実行するプロセッサに通知する．
- acquire：release を実行したプロセッサからの通知があるまで，acquire 以降のメモリ・アクセスの完了を（それがクリティカル・セクション内のメモリ・アクセスか否かに関わらず）遅らせる．

本来は，クリティカル・セクション内のメモリ・アクセスの完了のみを待ったり（*RC* の release の場合），あるいは，遅らせたり（*RC* の acquire の場合）すればよい．しかし，*RC* の現モデルでは，上記のように，クリティカル・セクション外のメモリ・アクセスの完了も待ったり（条件B-7），あるいは，遅らせたり（条件B-6）している．すなわち，オーバラップ実行可能なメモリ・アクセスを制限してしまっている．

そこで，以下のように，acquire および release が関与すべきメモリ・アクセスをクリティカル・セクション内のメモリ・アクセスのみに限定してみよう（図3参照）．

- release：対応するクリティカル・セクション内の先行するすべてのメモリ・アクセスが完了したことを，acquire を実行するプロセッサに通知する．
- acquire：release を実行したプロセッサからの通知があるまで，acquire 以降の対応するクリティカル・セクション内のメモリ・アクセスの完了を遅らせる．

本稿では，*IRC* に上記のアプローチを適用して得られる，以下のメモリ・コンシステンシ・モデルを新たに提案する．

① *Less Protective Consistency* (*LPC*)

5.3 提案モデルの定義

表2に，本稿で提案する4つのメモリ・コンシステンシ・モデル（*IWC*, *IRC*, *IDRF1*, *LPC*）に対して，3章で与えたフレームワークを適用した結果を示す．

以下，表中に現れ，かつ，4章で定義していない記号の定義を与える．

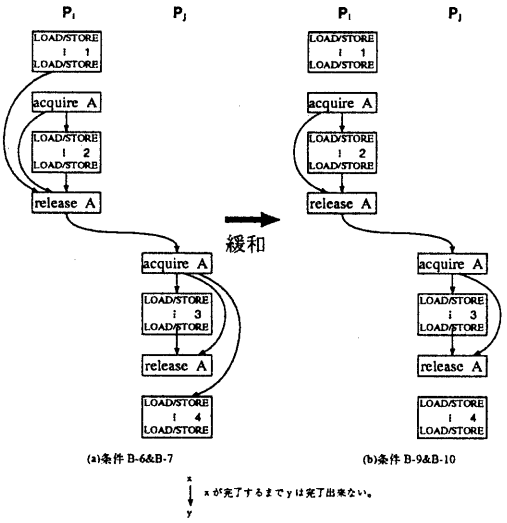

図 3: 条件 B-6&7 の緩和

5.3.1 メモリ・アクセス命令

表 2の I欄には，以下のメモリ・アクセス命令が加わる．

- Dload/ Dstore：プロセッサ内データ依存関係を保証する load/store.
- IDload/ IDstore：プロセッサ内データ依存関係を保証しない load/store.

5.3.2 オーダー

表 2の O欄に現れるオーダーを定義する前に，CS対 (*Critical-Section pair*)，および，2つのオーダー $\xrightarrow{ao}$ と $\xrightarrow{ro}$[10] を定義する．

定義 8 (CS 対) [3]
メモリ・アクセス集合の 2 つの要素 $M_1 = (P_1, N_1, T_1, L_1)$ と $M_2 = (P_2, N_2, T_2, L_2)$ が以下の条件を満たす時，「M_1と M_2は CS 対 (*Critical-Section pair*) を成す」と言う．

- $P_1 = P_2$
- $N_1 < N_2$
- T_1 = acquire
- T_2 = release
- $L_1 = L_2$
- 次の条件を満たすメモリ・アクセス $M_3 = (P_3, N_3, T_3, L_3)$ が存在しない．
  - $P_3 = P_1 (= N_2)$
  - $N_1 < N_3 < N_2$
  - T_3 = acquire または release
  - $L_3 = L_1 (= L_2)$

[10] これらのオーダーは，表 2中に直接は現れない．

定義 9 (acquire-order と release-order) [3]
メモリ・アクセス集合の 3 つの要素 $M_1 = (P_1, N_1, T_1, L_1)$，$M_2 = (P_2, N_2, T_2, L_2)$ と $M_3 = (P_3, N_3, T_3, L_3)$ が以下の条件を満たす時，および，その時に限り M_1と M_3の間に $M_1 \xrightarrow{ao} M_3$なるオーダー「$\xrightarrow{ao}$」が，また，M_2と M_3の間に $M_3 \xrightarrow{ro} M_2$なるオーダー「$\xrightarrow{ro}$」が存在する．

- M_1と M_2が CS対を成す．
- $P_1 = P_2 = P_3$
- $N_1 < N_3 < N_2$

表 2の O欄には，以下のオーダーが加わる．

- $\xrightarrow{po1}$ (*program-order-1*)：メモリ・アクセス集合の 2 つの要素 $M_1 = (P_1, N_1, T_1, L_1)$ と $M_2 = (P_2, N_2, T_2, L_2)$ が以下の条件を満たす時，および，その時に限り M_1と M_2の間に $M_1 \xrightarrow{po1} M_2$ なるオーダー「$\xrightarrow{po1}$」が存在する．
  - $P_1 = P_2$
  - $N_1 < N_2$
  - T_2 = Dload または Dstore
- $\xrightarrow{po2}$ (*program-order-2*)：メモリ・アクセス集合の 2 つの要素 $M_1 = (P_1, N_1, T_1, L_1)$ と $M_2 = (P_2, N_2, T_2, L_2)$ が以下の条件を満たす時，および，その時に限り M_1と M_2の間に $M_1 \xrightarrow{po2} M_2$ なるオーダー「$\xrightarrow{po2}$」が存在する．
  - $P_1 = P_2$
  - $N_1 < N_2$
  - 以下の条件を満たすメモリ・アクセス集合の要素 $M_3 = (P_3, N_3, T_3, L_3)$ が少なくとも 1 つ存在する．
    - $P_1 = P_2 = P_3$
    - $N_1 \leq N_3 \leq N_2$
    - T_3 = special
- $\xrightarrow{pso3}$ (*program-special-order-3*)：

$$\xrightarrow{pso3} = (\xrightarrow{po0})^* \xrightarrow{so0} (\xrightarrow{po0} \cup \xrightarrow{so0})^*$$

- $\xrightarrow{pso4}$ (*program-special-order-4*)：

$$\xrightarrow{pso4} = (\xrightarrow{po0})^* \xrightarrow{so1} (\xrightarrow{po0} \cup \xrightarrow{so1})^*$$

- $\xrightarrow{pso5}$ (*program-special-order-5*)：

$$\xrightarrow{pso5} = (\xrightarrow{po0})^* \xrightarrow{so2} (\xrightarrow{po0} \cup \xrightarrow{so2})^*$$

- $\xrightarrow{pso6}$ (*program-special-order-6*)：

$$\xrightarrow{pso6} = (\xrightarrow{ro} \cup \phi) \xrightarrow{so2} (\xrightarrow{po0} \cup \xrightarrow{so2})^* (\xrightarrow{ao} \cup \phi)$$

5.3.3 ハードウェアが遵守すべき条件

表 2の P_{HW} 欄には，以下の条件が加わる．

- 条件 A — プロセッサ内データ依存関係の保証：

- 条件 A-1：プロセッサ内データ依存関係を保証する．すなわち，$M_1 \xrightarrow{pol} M_2$なる衝突対について，$M_2$をプロセッサ P_2（$= P_1$）に関して完了させるためには，M_1がプロセッサ P_1（$= P_2$）に関して完了していなければならない．

- 条件 B — メモリ・アクセスを実行したプロセッサが当該メモリ・アクセスを完了させるための条件：
  - 条件 B-9 (*Intraprocessor, Acquire → Ordinary, Restricted*)：あるプロセッサが実行した load/store$M_3 = (P_3, N_3, T_3, L_3)$ を完了させるためには，当該プロセッサがそれ以前に実行した以下の条件を満たすすべての acquireM_1が完了していなければならない．
    - $M_1 = (P_1, N_1, T_1, L_1)$ と $M_2 = (P_2, N_2, T_2, L_2)$ とが *CS* 対を成す．
    - $N_1 < N_3 < N_2$
  - 条件 B-10 (*Intraprocessor, Ordinary → Release, Restricted*)：あるプロセッサが実行した release$M_2 = (P_2, N_2, T_2, L_2)$ を完了させるためには，当該プロセッサがそれ以前に実行した以下の条件を満たすすべての load/storeM_3が完了していなければならない．
    - $M_1 = (P_1, N_1, T_1, L_1)$ と $M_2 = (P_2, N_2, T_2, L_2)$ とが *CS* 対を成す．
    - $N_1 < N_3 < N_2$

- 条件 C — メモリ・アクセス完了の順序に関する条件：
  - 条件 C-3：$M_1 \xrightarrow{pso4} M_2$なる衝突対を成すメモリ・アクセス M_1および M_2は，すべてのプロセッサに関して M_1，M_2の順序で完了しなければならない．

6 おわりに

以上，メモリ・コンシステンシ・モデルに対して統一的枠組（フレームワーク）を与え，5つの従来モデル（*SC, WC, DRF0, RC, DRF1*）に適用した．さらに，新しい4つのモデル（*IWC, IRC, IDRF1, LPC*）を提案した．

今後の課題としては，以下のものを考えている．

- 各メモリ・コンシステンシ・モデルの潜在能力の定式化および比較
- 各モデルに対する各種実現方法の考案，および，その理論性能ならびにハードウェア・コストの定式化および比較
- シミュレータあるいは実機を用いての実効性能の評価

謝辞

日頃ご討論頂くとともに貴重なご示唆を与えて下さった，九州大学 大学院総合理工学研究科 安浦寛人教授に深謝致します．九州大学並列処理&計算研究会第1回研究会（*PLD-1: 1st Parallel-Land Day*）において貴重なコメントを頂戴した，九州大学 工学部 岩間一雄 教授，同 荒木啓二郎 助教授，および，同 福田 晃 助教授に感謝します．最後に，安浦研究室の諸氏に感謝致します．

本研究は一部，文部省科学研究費補助金 重点領域研究「超並列原理に基づく情報処理体系」による．

参考文献

[1] 徳永尚哉, 村上和彰, 山家 陽, "高性能プラットフォーム要素マイクロプロセッサ・アーキテクチャ— 通信モデルに関する検討 —," 信学技報, CPSY-92-20, 1992年8月.

[2] 徳永尚哉, "メモリ・コンシステンシ・モデル— フレームワーク化 —," 九州大学 大学院総合理工学研究科 情報システム学専攻情報組織講座, 第12回研究会研究発表資料, TR-92-20, 1992年10月.

[3] 徳永尚哉, 村上和彰, "メモリ・コンシステンシ・モデル— フレームワーク化および新モデルの提案 —," 九州大学並列処理&計算研究会, 第1回研究会資料, 1992年11月.

[4] Adve, S. V. and Hill, M. D., "Weak Ordering – A New Definition," *Proc. 17th Int'l. Symp. Computer Architecture*, pp.2–14, May 1990.

[5] Adve, S. V. and Hill, M. D., "A Unified Formalization of Four Shared-Memory Models," Computer Sciences Technical Report # 1051, Univ. of Wisconsin, Madison, Sept. 1991.

[6] Dubois, M., Scheurich, C., and Briggs, F. A., "Memory Access Buffering in Multiprocessors," *Proc.13th Int'l. Symp. Computer Architecture*, pp.434–442, June 1986.

[7] Gharachorloo, K., Lenoski, D., Laudon, J., Gibbons, P., Gupta, A., and Hennessy, J., "Memory Consistency and Event Ordering in Scalable Shared-Memory Multiprocessors," *Proc.17th Int'l. Symp. Computer Architecture*, pp.15–26, May 1990.

[8] Gharachorloo, K., Gupta, A., and Hennessy, J., "Performance Evaluation of Memory Consistency Models for Shared-Memory Multiprocessors," *Proc. 4th Int'l Conf. Architectural Support for Programming Languages and Operating Systems*, pp.245–257, Apr. 1991.

[9] Goodman, J. R., "Cache consistency and sequential consistency," Computer Sciences Technical Report # 1006, Univ. of Wisconsin, Madison, Feb. 1991.

[10] Lamport, L., "How to Make a Multiprocessor Computer That Correctly Executes Multiprocess Programs," *IEEE Trans. Comput.*, vol.C-28, no.9, pp.690–691, Sept. 1979.

[11] Shasha, D. and Snir, M., "Efficient and Correct Execution of Parallel Programs that Share Memory," *ACM Trans. Programming Languages and Systems*, vol.10, no.2, pp.282–312, Apr. 1988.

表 1: 従来のメモリ・コンシステンシ・モデルの 4 つ組

CM	I					O										P_{SW}		P_{HW}														
	load/store		special			$\xrightarrow{po}$	$\xrightarrow{so}$			$\xrightarrow{pso}$								A	B									C				
	D	ID	acq	rel	nsy	0	0	1	2	0	1	2						1	2	0	0	1	2	3	4	5	6	7	8	0	1	2	
SC	o		–			o	–	–	–	–	–	–						o	–	√	o	–	–	–	–	–	–	–	–	√	–	–	
WC	o		o			o	o	√	√	o	√	√						√	o	o	×	o	o	o	√	√	√	√	×	×	×	√	
$DRF0$	o		o			o	o	√	√	o	√	√						√	o	o	×	×	×	×	o	o	×	×	×	o	o	√	
RC	o		o	o	o	o	o	√	o	×	√	o						√	o	o	×	×	×	o	×	×	o	o	×	×	×	√	
$DRF1$	o		o	o	o	o	o	o	√	×	o	×						√	o	o	×	×	×	o	×	×	o	×	o	√	√	o	

D : Dload/Dstore
ID : IDload/IDstore
acq : acquire
rel : release
nsy : nsync

o : モデルの定義に用いる要素
√ : o に包含される要素
× : モデルの定義に用いてはならない要素
– : モデルの定義に無関係な要素

表 2: 提案するメモリ・コンシステンシ・モデルの 4 つ組

CM	I					O														P_{SW}		P_{HW}																
	load/store		special			$\xrightarrow{po}$			$\xrightarrow{so}$			$\xrightarrow{pso}$										A	B												C			
	D	ID	acq	rel	nsy	0	1	2	0	1	2	0	1	2	3	4	5	6	1	2	0	1	0	1	2	3	4	5	6	7	8	9	10	0	1	2	3
IWC	o	o		o		×	o	o	o	√	√	×	×	×	o	√	√	√	√	o	×	o	×	o	o	o	√	√	√	√	×	√	√	×	×	×	√
IRC	o	o	o	o	o	×	o	×	o	√	√	×	×	×	×	√	o	√	√	o	×	o	×	×	×	o	×	×	o	o	×	√	√	×	×	×	√
$IDRF1$	o	o	o	o	o	×	o	×	o	√	√	×	×	×	×	o	×	×	√	o	×	o	×	×	×	o	×	×	o	×	o	√	×	×	√	×	o
LPC	o	o	o	o	o	×	o	×	o	√	√	×	×	×	×	×	×	o	√	o	×	o	×	×	×	o	×	×	×	×	×	o	o	×	×	×	×

D : Dload/Dstore
ID : IDload/IDstore
acq : acquire
rel : release
nsy : nsync

o : モデルの定義に用いる要素
√ : o に包含される要素
× : モデルの定義に用いてはならない要素
– : モデルの定義に無関係な要素